
Performance Impact of Some WebLogic JDBC Parameters

Ken Gottry
ken@gottry.com
Aug-2003

Objective

- JDBC testing has shown that setting WebLogic parameters to continually test the health of JDBC entries
 - Degrades response time
 - Improves failure detection and recovery
- This presentation attempts to
 - Quantify the degradation of response time . . . based on infra testing without an application
 - Describe the impact of these WebLogic parameters on failure detection and recovery

Test Environment

- Hardware
 - **Server:** SunFire 280R with two 900MHz CPU's and 8GB memory
 - **Client:** Windows 2000 Professional with one 1.2GHz CPU and 512MB memory
 - **Network:** 100MB
- WebLogic *ExecuteThreads=15*
- JDBC Pool *InitialCapacity=MaxCapacity=130*
- Simulated 15 virtual users using Grinder (<http://sourceforge.net/projects/grinder/>)
 - Each user repeated a test script for 2 minutes
 - The test script consists of 10 requests for the JSP with 400ms pause between each request
- The JSP issued the following SQL statement that returned 300 rows

```
select user_name, user_num from user_table
```
- Two JSP's were used
 - Rapid Fire
 - JSP obtains a JDBC connection, issues the Select, then ends
 - Sleep to simulate processing
 - JSP obtains a JDBC connection, issues the Select, **sleeps for 1 second**, then ends
- The Simulate Processing tests resulted in the virtual users submitting requests at a slower rate

Response Time Analysis

TestConnectionsOnReserve

TestConnectionsOnReserve

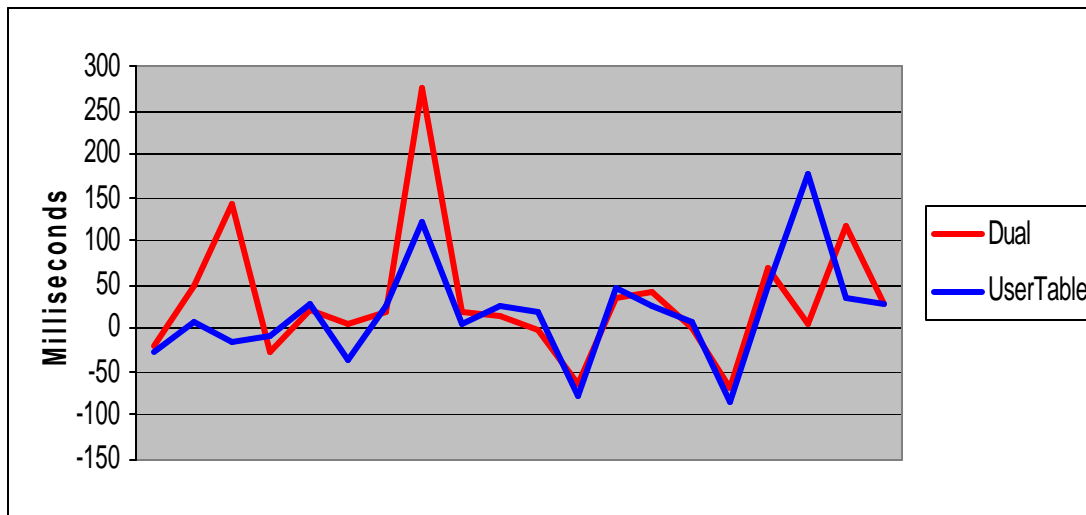
- When this parameter is set to *true* in the JDBC Pool definition, WebLogic will test a JDBC entry before handing it to an application for use.
- WebLogic issues a *select count(*) from TestTable* SQL call to see if the JDBC entry is alive and functioning

Other related parameters are:

- *TestConnectionsOnRelease* – When the application closes the JDBC connection, WebLogic will test the JDBC entry before returning it to the JDBC Pool
- *RefreshMinutes* – This parameter defines an interval following which WebLogic will test every unused connection. Connections that do not pass the test will be closed and recreated

Rapid Fire Test

- The ZERO line on the Y-axis represents the response time without TestConnectionsOnReserve
- The red and blue lines show the increase or decrease in response time when *TestConenctionsOnReserve=true*.
- WebLogic issues this command to test a connection
`Select count(*) from TestTableName`
- The red line represents when *TestTableName=dual* while the blue line represents when *TestTableName=User_Table* (an application table)

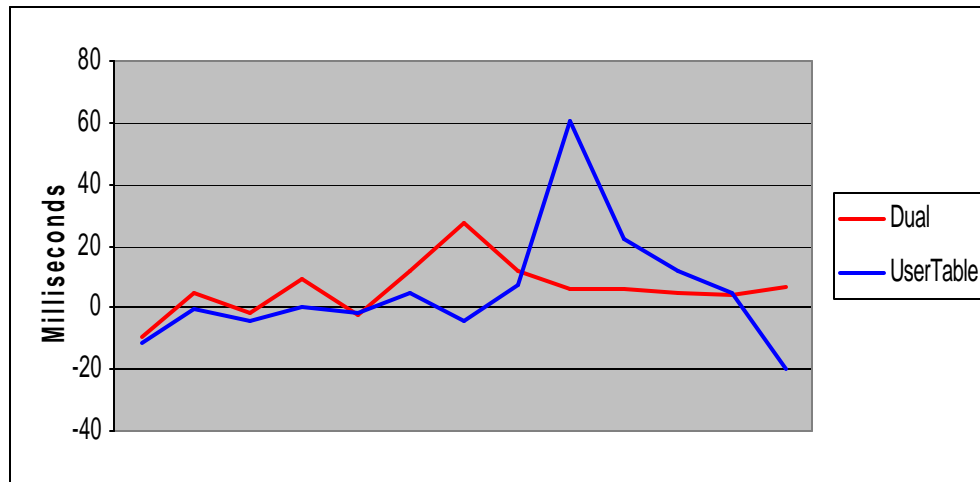


Conclusions

- TestConnectionsOnReserve adds 50-100ms
- User table performed better than Dual table

Tests to Simulate Processing

- The ZERO line on the Y-axis represents the response time without *TestConnectionsOnReserve*
- The red and blue lines show the increase or decrease in response time when *TestConnectionsOnReserve*=true.
- WebLogic issues this command to test a connection
`Select count(*) from TestTableName`
- The red line represents when *TestTableName*=dual while the blue line represents when *TestTableName*=User_Table (an application table)



Conclusions

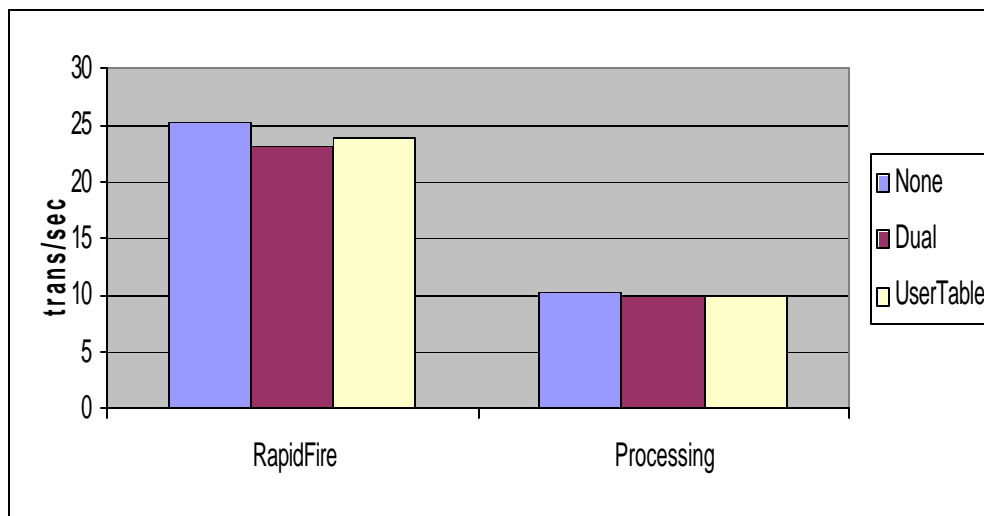
- *TestConnectionsOnReserve* adds 10-30ms
- User table performed better than Dual table

Supposition

The *TestConnectionsOnReserve* delay is less when the requests come at a slower rate

Impact on Transaction/Second (TPS)

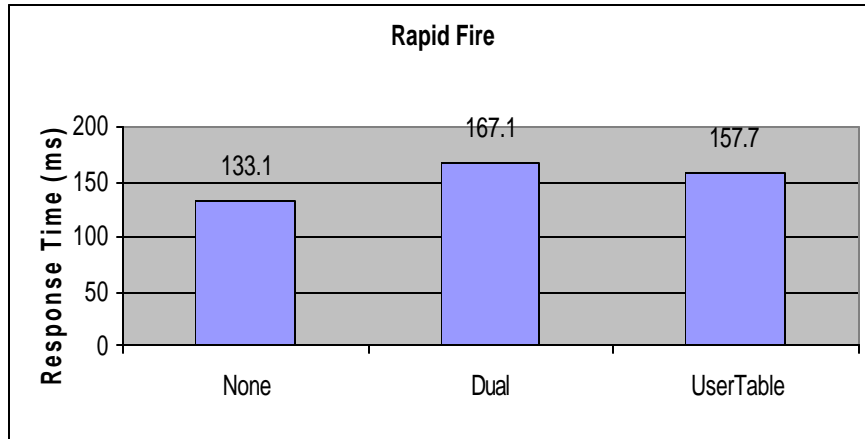
- As expected the TPS rate was much lower for the Simulate Processing tests (when the JSP slept for 1 second after issuing the Select statement) than for the Rapid Fire tests
- With the higher request rate of the Rapid Fire test, there was a 5-8% difference in the TPS based on different TestOnReserve options used
- With the lower request rate of the Simulate Processing test, there was only a 0.5-1% difference in the TPS based on different TestOnReserve options used



Supposition

The impact of TestOnReserve is lower when the requests come at a slower rate

Impact on Response Time



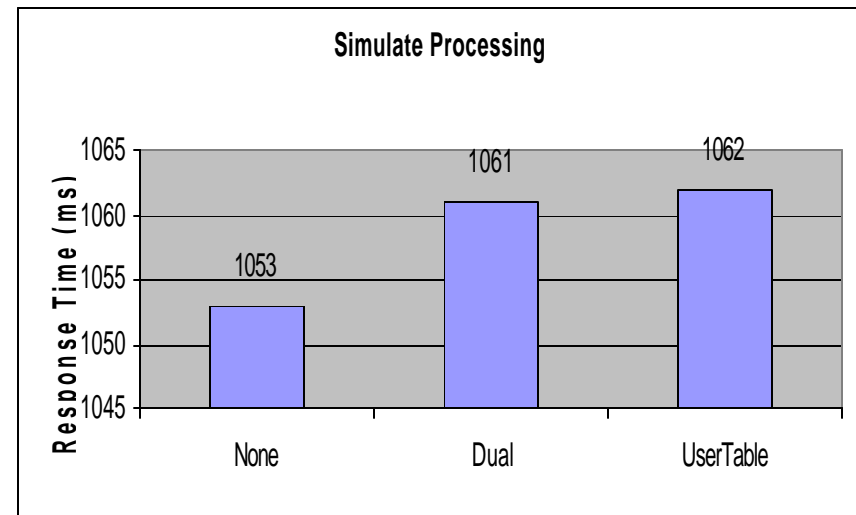
Dual added 25% to response time
User_Table added 18% to response time



Dual added less than 1% to response time
User_Table less than 1% to response time



- **None** means *TestConnectionsOnReserve=false*
- **Dual** means *TestConnectionsOnReserve=true* and *TestTableName=dual*
- **LoginUser** means *TestConnectionsOnReserve=true* and *TestTableName=User_Table*



Failure Detection and Recovery

Failure Detection and Recovery – Test 1

Test 1

- Started WebLogic with *InitialCapacity=10* and *TestConnectionsOnReserve=false*.
- `netstat -na | grep 1521` showed 10 ESTABLISHED connections between WebLogic and Oracle
- Stopped Oracle
- `netstat -na | grep 1521` showed 10 CLOSE-WAIT connections indicating the connections to Oracle had been closed
- Restarted Oracle
- `netstat -na | grep 1521` showed ZERO connections between WebLogic and Oracle
- Submitted a JSP request that required a database connection. It timed out and failed
- Waited the number of minutes specified in *RefreshMinutes* parameter
- `netstat -na | grep 1521` showed ZERO connections between WebLogic and Oracle
- Submitted a JSP request that required a database connection. It still failed.

Failure Detection and Recovery – Test 2

Test 2

- Started WebLogic with *InitialCapacity=10* and *TestConnectionsOnReserve=true*.
- `netstat -na | grep 1521` showed 10 ESTABLISHED connections between WebLogic and Oracle
- Stopped Oracle
- `netstat -na | grep 1521` showed 10 CLOSE-WAIT connections indicating the connections to Oracle had been closed
- Restarted Oracle
- `netstat -na | grep 1521` showed ZERO connections between WebLogic and Oracle
- Submitted one JSP request that required a database connection. It worked
- `netstat -na | grep 1521` showed 10 ESTABLISHED connections between WebLogic and Oracle

Conclusions

- It appears that WebLogic creates JDBC pools
 - at WebLogic startup
 - when WebLogic detects that no pool exists (or not enough entries exist in a pool)
- In Test #1, it appears that WebLogic believes the JDBC pool exists but that all ZERO of the ZERO entries are in use, therefore the next JSP request fails with a connection timeout
- In Test#2, it appears the *TestConnectionsOnReserve* setting causes WebLogic to realize that a valid connection is not available. This realization appears to cause WebLogic to try to (re)build the JDBC pool, which results in new JDBC entries being created.

Conclusions - continued

- When Oracle is stopped and restarted (either in a controlled fashion or due to a database failure), there are four ways to make WebLogic detect this and rebuild its JDBC pool :
 - Use the *TestConnectionsOnReserve* parameter in the JDBC definition
 - Use the *RefreshMinutes* parameter in the JDBC definition
 - Execute the *weblogic.Admin* command line utility with the *Reset_Pool* option
 - Stop and restart WebLogic
- Pros and cons of each option
 - Using the *TestConnectionsOnReserve* parameter adds 10-100ms to every SQL call.
 - Using the *RefreshMinutes* parameter appears to require the *TestConnectionsOnReserve* parameter also
 - Using the *weblogic.Admin* utility requires some process external to WebLogic to detect the failure and initiate the utility. Also, since the *Reset_Pool* command closes all connections and rebuilds a new JDBC pool, its use can be intrusive to a functioning WebLogic server
 - Using the stop/restart option causes the outage of the WebLogic server